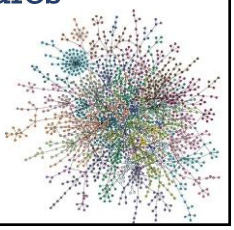
CSCI 262
Data Structures

22 – Graphs

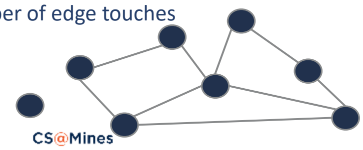


Today's Lecture

- Describe the properties, relationships among vertices and edges, and types (e.g., simple, complete).
- Discuss examples and possible applications of various kinds of graphs
- Identify two principal data structures for graphs (e.g., adjacency matrix and adjacency lists)
- Compare graph traversal techniques (e.g., breadth-first and depth-first)

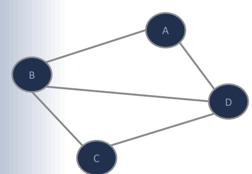
Graphs

- Model the ***relationships*** between ***things***
- Composed of *vertices* and *edges.*
- We write G = (V,E)
  - V = set of vertices (aka nodes or *things*)
  - E = set of edges (aka relationships)
  - Degree = number of edge touches

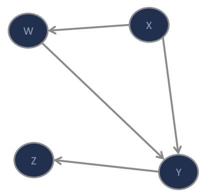


Types of Graphs

Undirected Graph
Edges are bi-directional



Directed Graph

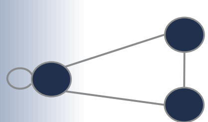


Paths in graphs: a path is a sequence of vertices p_0, p_1, …, p_m such that there is an edge from p_i to p_{i+1}. We say there is a path from p_0 to p_m.

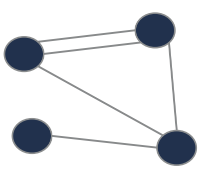
Above, there is a path from A to C and C to A, and from W to Z but not from Z to W.

Simple Graphs

Non-simple:
Has *self edges* aka *loops*

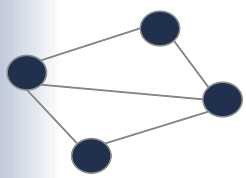


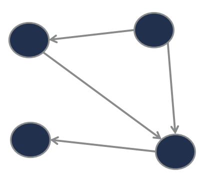
Non-simple:
Multiple edges



We will be primarily concerned with simple graphs, i.e., ones with no self edges and no multiple edges.

Some Applications

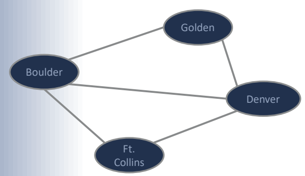




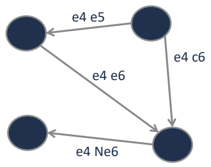
Q. What kinds of graphs do you encounter daily?

Some Applications

Route Finding



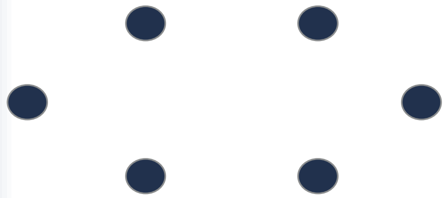
Game State Graph
(Bogus Chess Example)



Many other applications – consider just *some* of the graphs we interact with:

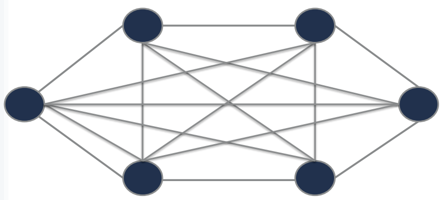
- Networks (electrical, digital, phone)
- Flows (natural resources such as water, supply chain, traffic)
- Assemblies (molecules, circuit boards, software modules)

Exercise – Complete Graph



Definition – A complete graph is where every pair of vertices (V) in a simple undirected graph (G) is connected by a unique edge (E).
Vertices = n = Degree = n-1= # Edges =

Exercise – Complete Graph



Definition – A complete graph is where every pair of vertices (V) in a simple undirected graph (G) is connected by a unique edge (E).
Vertices = n = 6 Degree = n-1= 5 # Edges = 15

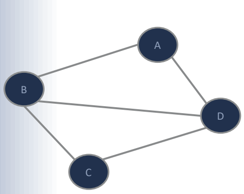
n(n-1)

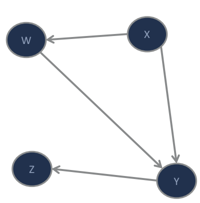
2

Data Structures for Graphs

- Two principal data structures:
  - Adjacency matrix
  - Adjacency lists
- Applications for each, but:
  - Adjacency matrix always large: $|V|^2$
  - Adjacency lists often more efficient
    - Only stores edges that exist
    - Most graphs are *sparse* – have $|E| << |V|^2$

Adjacency Matrix

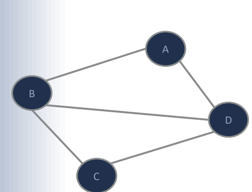


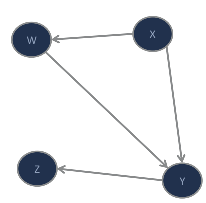


	A	B	C	D
A	0	1	0	1
B	1	0	1	1
C	0	1	0	1
D	1	1	1	0

	W	X	Y	Z
W	0	0	1	0
X	1	0	1	0
Y	0	0	0	1
Z	0	0	0	0

Adjacency Lists





A	B	C	D
B	A	B	A
D	C	D	B
	D		C

Typically, linked lists

W	X	Y	Z
Y	W	Z	
	Y		

Graph Traversal (Search)

Two principal ways of traversing a graph:

- Depth First Search (DFS)
  - Start at some vertex
  - Follow a simple path discovering new vertices until you cannot find a new vertex.
  - Back-up until you can start finding new vertices.
- Breadth First Search (BFS)
  - Starting at a source vertex
  - Explores the edges to "discover" every vertex reachable from the source.

CS@Mines

Depth First Search (Recursive)

```
// initialization
for all u in V:
   set u.visited = false

// Traverse graph G starting from node v
dfs(G, v)
   set v.visited = true
   for each edge (v,u) in E:
      if not u.visited
         do dfs(G, u)
```

CS@Mines

Depth First Search (Stack)

```
dfs(G, v)
   for all u in V:
      set u.visited = false

   let S be a stack
   set v.visited = true
   S.push(v)
   while S not empty:
      u = S.pop()
      for all edges (u, w) in E:
         if not w.visited:
            S.push(w)
            set w.visited = true
```

CS@Mines

Breadth First Search (Queue)

```
bfs(G, v)
   for all u in V:
      set u.visited = false

   let Q be a queue
   set v.visited = true
   Q.push(v)
   while Q not empty:
      u = Q.pop()
      for all edges (u, w) in E:
         if not w.visited:
            Q.push(w)
            set w.visited = true
```

CS@Mines

Other Algorithms to Explore

- Route finding (shortest/best paths)
  - Dijkstra's algorithm
  - A*
- Minimum Spanning Tree – Connect a graph using the least resources (edge weights)
  - Kruskal's algorithm
  - Prim's algorithm
- Max flow – what is the maximum amount you can move along a network?
- Game playing
  - Minimax
  - Alpha-beta pruning, iterative deepening, many more

You can find all of these on Wikipedia…

CS@Mines

Up Next

- Wednesday, May 2
  - Final exam review
  - Project 5 due
- Thursday, May 10
  - 8:00 am – 10:00 am: Section A final (BB W280)
  - 3:15 pm – 5:15 pm: Section B final (CO 209)

CS@Mines